Unit 2:
Objects and Classes

What we will learn

- ✓ Defining classes for objects
- ✓ Constructors
- ✓ Accessing objects via reference variable
- ✓ Using classes from the java library
- ✓ Static variables, constants and methods
- ✓ visibility modifiers
- ✓ Data field encapsulation
- ✓ Passing objects to methods, array of objects,
- ✓ Immutable objects and classes,
- ✓ scope of variable
- ✓ this reference.

Unit-4

Introduction

Large Data Handling

Class

- Class is **derived datatype**, it combines members of different datatypes into one.
- Defines new datatype (primitive ones are not enough).
  - ➥ For Example : **Car, College, Bus etc..**
- This new datatype can be used to create objects.
- A class is a template for an object .

Example :

```
class Car{
        String company;
        String model;
        double price;
        double milage;
        .........
}
```

Car Class

Class: Car

Properties (Describe)
Company
Model
Color
Mfg. Year
Price
Fuel Type
Mileage
Gear Type
Power Steering
Anti-Lock braking system

Methods (Functions)
`Start`
`Drive`
`Park`
`On_break`
`On_lock`
`On_turn`

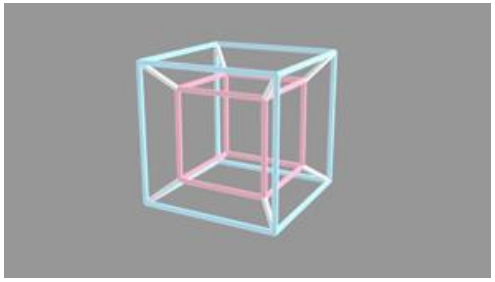
Object

- An object is an **instance** of a **class**.
- An object has a **state** and **behavior**.

  Example: A dog has

  states - color, name, breed as well as

  behaviors – barking, eating.
- The **state** of an object is stored in **fields** (variables), while **methods** (functions) display the object's **behavior**.



What is an Object?

Philosophy of Object Oriented

- Our real world is nothing but classification of objects
  - E.g. Human, Vehicle, Library, River, Watch, Fan, etc.
- Real world is organization of different objects which have their own characteristics, behavior
  - Characteristic of Human: Gender, Age, Height, Weight, Complexion, etc.
  - Behavior of Human: Walk, Eat, Work, React, etc.
  - Characteristic of Library: Books, Members, etc.
  - Behavior of Library: New Member, Issue Book, Return Book etc.

What is an Object?

Pen

Board

Laptop

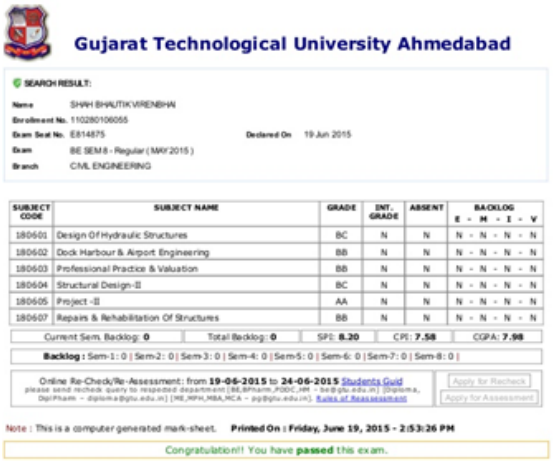
Bench

Projector

Bike

Physical objects…

What is an Object? (Cont…)



Gujarat Technological University Ahmedabad

SEARCH RESULT:

Name SHAH BHAUTIK VIRENBHAI
Enrollment No. 110280106055
Exam Seat No. E814875 Declared On 19-Jun-2015
Exam BE SEM-8 - Regular (MAY 2015)
Branch CIVIL ENGINEERING

SUBJECT CODE	SUBJECT NAME	GRADE	INT. GRADE	ABSENT	BACKLOG E - M - I - V
180601	Design Of Hydraulic Structures	BC	N	N	N - N - N - N
180602	Dock Harbour & Airport Engineering	BB	N	N	N - N - N - N
180603	Professional Practice & Valuation	BB	N	N	N - N - N - N
180604	Structural Design-II	BC	N	N	N - N - N - N
180605	Project -II	AA	N	N	N - N - N - N
180607	Repairs & Rehabilitation Of Structures	BB	N	N	N - N - N - N

Current Sem. Backlog: 0 | Total Backlog: 0 | SPI: 8.20 | CPI: 7.58 | CGPA: 7.98

Backlog : Sem-1 : 0 | Sem-2 : 0 | Sem-3 : 0 | Sem-4 : 0 | Sem-5 : 0 | Sem-6 : 0 | Sem-7 : 0 | Sem-8 : 0 |

Online Re-Check/Re-Assessment : from 19-06-2015 to 24-06-2015 Students Guid
please send recheck query to respected department [BE,BPharm,PODC,MM - be@gtu.edu.in] [Diploma, DipPharm - diploma@gtu.edu.in] [ME,MPH,MBA,MCA - pg@gtu.edu.in]. Rules of Reassessment
Apply for Recheck
Apply for Assessment

Note : This is a computer generated mark-sheet. Printed On : Friday, June 19, 2015 - 2:53:26 PM

Congratulation!! You have passed this exam.

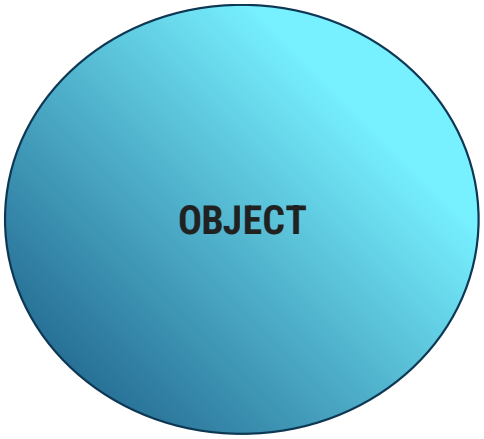
Result

Bank Account

Logical objects…

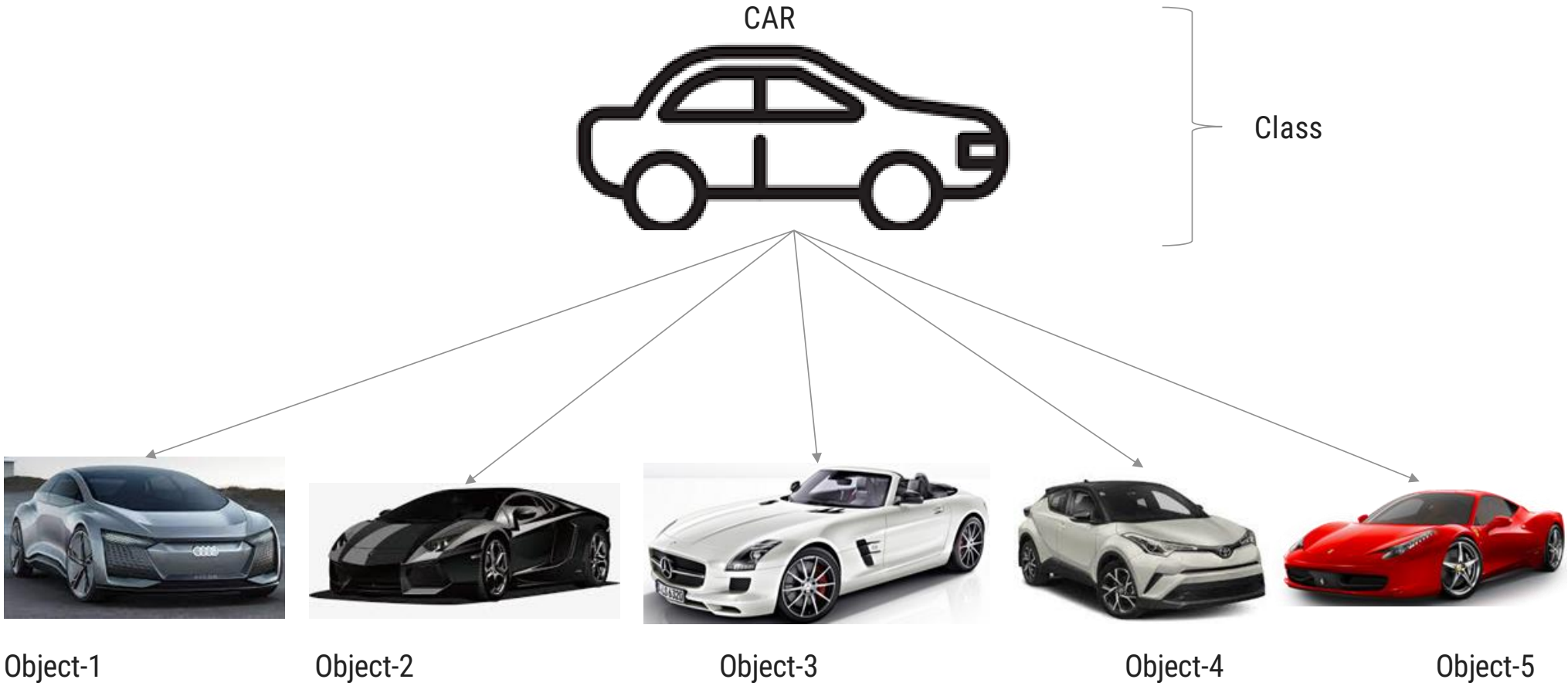
What is an Object?

- An Object is a key to understand Object Oriented Technology.
- An entity that has state and behavior is known as an object. e.g., Mobile, Car, Door, Laptop etc
- Each and every object posses
  - Identity
  - State
  - Behavior

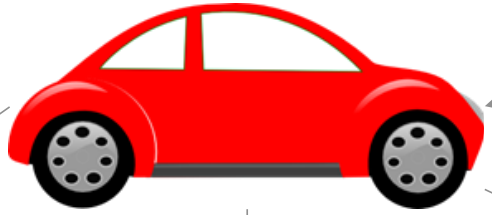


Object is an Instance of Class

Object: A Real-World Entity



Object: A Real-World Entity



OBJECT: CAR

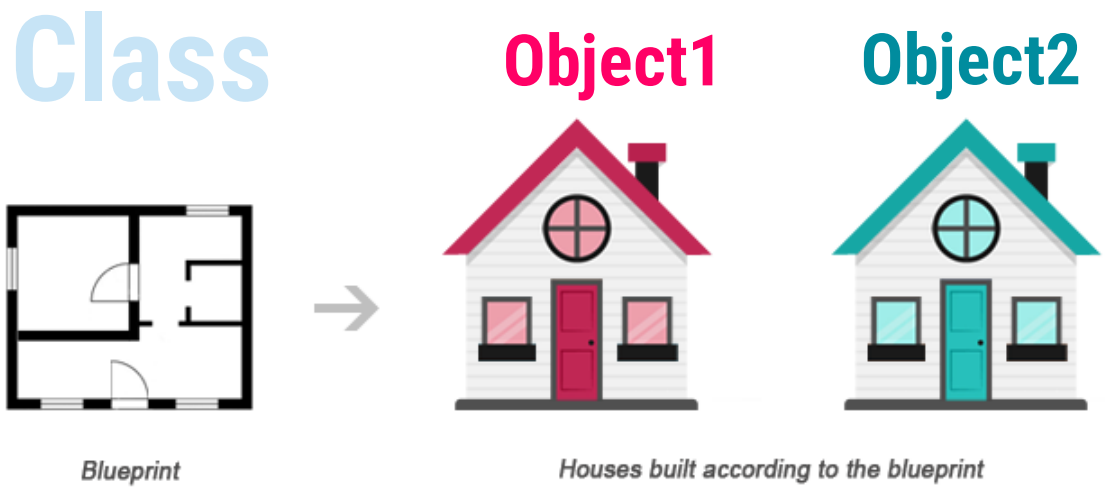
Properties (Describe)
Manufacturer
Model
Color
Year
Price

Methods (Actions)
Start
Drive
Park

Events
On_Start
On_Parked
On_Brake

Classes and Objects

Classes and Objects



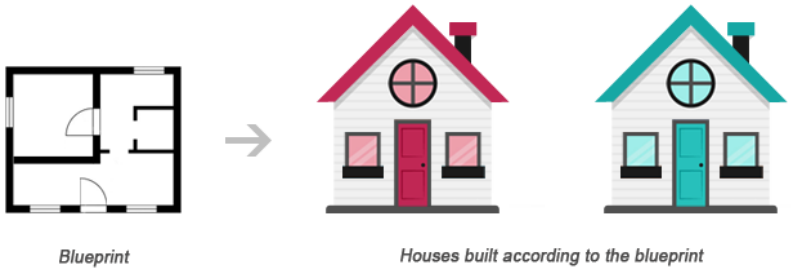
Class is a blueprint of an object

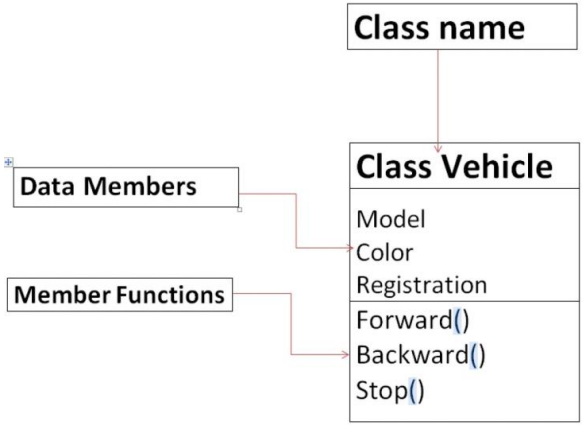
Class describes the object

Object is instance of class

What is Class?

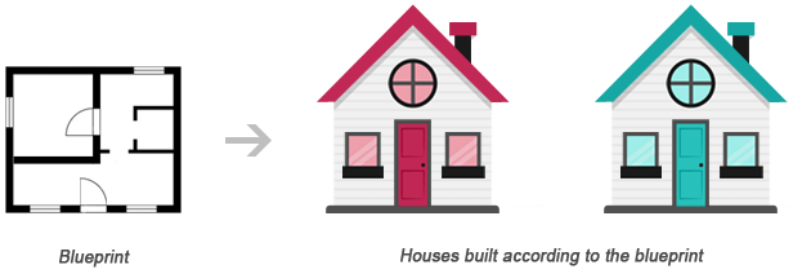
- **Class can be defined in multiple ways**
  - A class is the building block.
  - A class is a blueprint for an object.
  - A class is a user-defined data type.
  - A class is a collection of objects of the similar kind.
  - A class is a user-defined data type which combines data and methods.
  - A class describes both the data and behaviors of objects.
- Class contains data members (also known as field or property or data) and member functions (also known as method or action or behavior)
- Classes are similar to structures in C.
- Class name can be given as per the Identifier Naming Conventions.





What is Object?

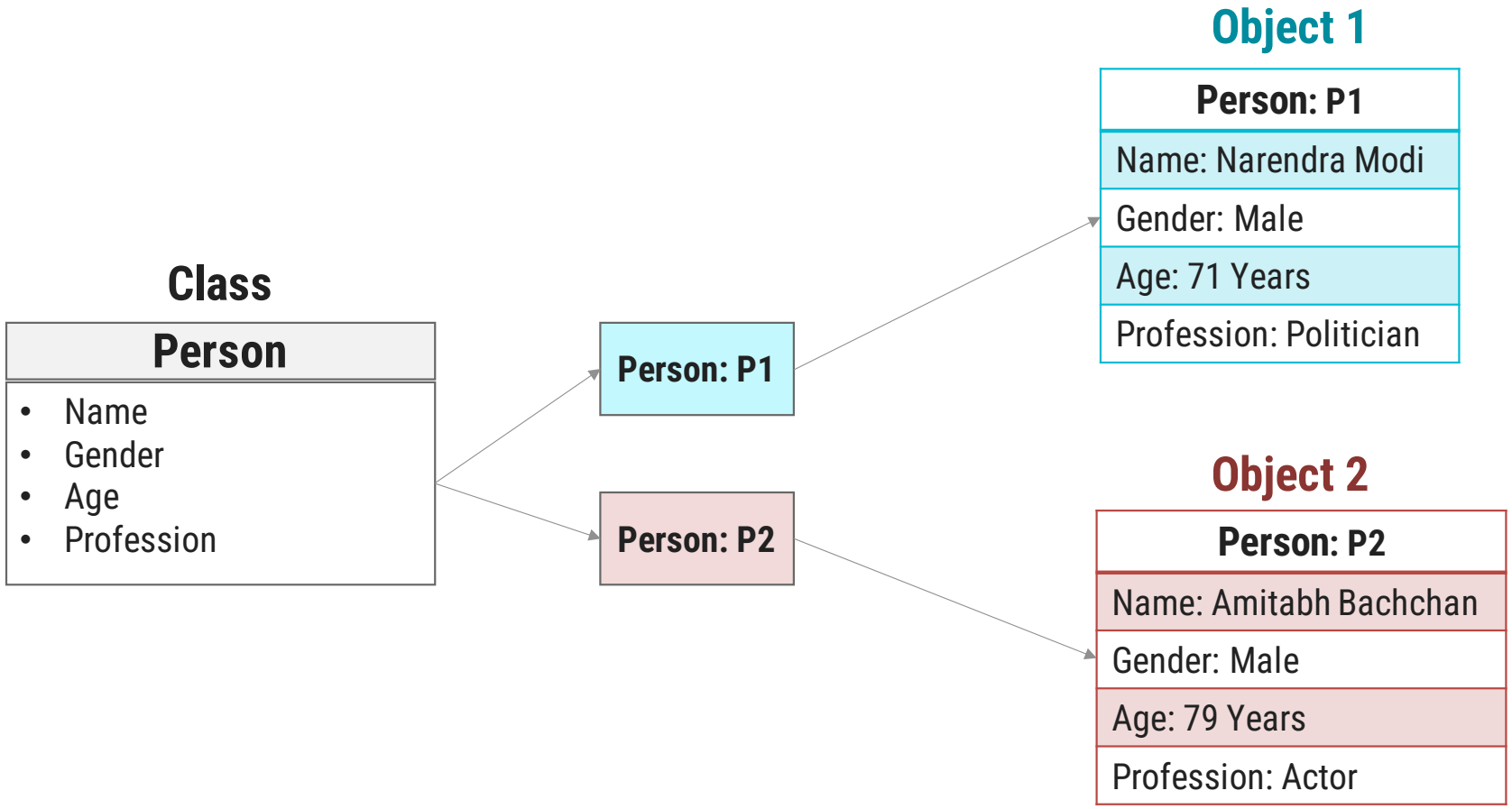
- **Definition**: An Object is an instance of a Class.
- An Object is a variable of a specific Class
- An Object is a data structure that encapsulates data and functions in a single construct.
- Object is a basic run-time entity
- Objects are analogous to the real-world entities.



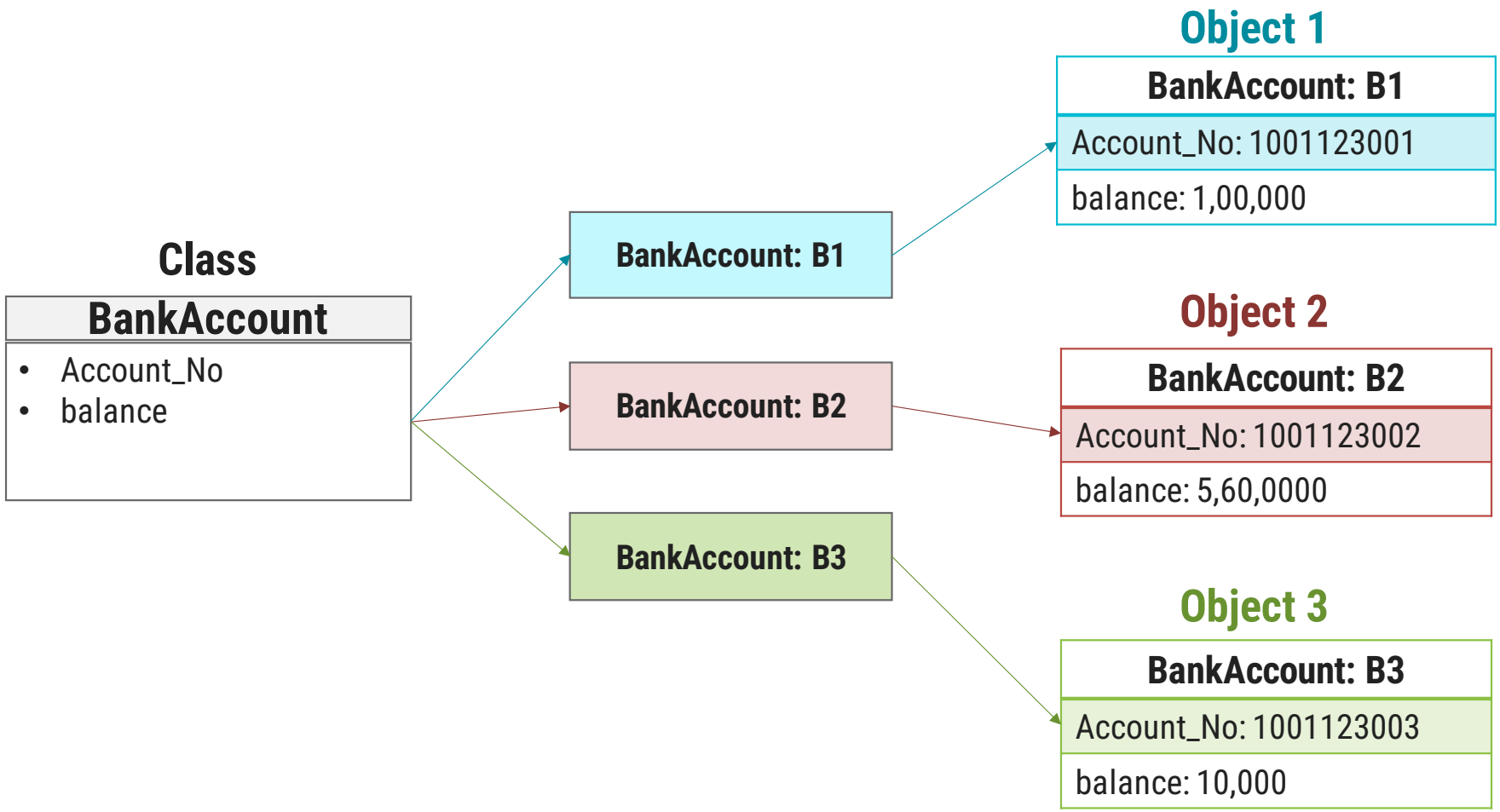
Points to Remember

- When a class is defined, only the specification or blueprint for the object is defined; no memory or storage is allocated.
- When an object of a class is declared, the memory is allocated as per the data members of a class
- We can access the data members and member functions of a class by using a . (dot) operator.
- Generally Class contains
  - Data Members
  - Member Functions
  - Constructor (Special Member Function)

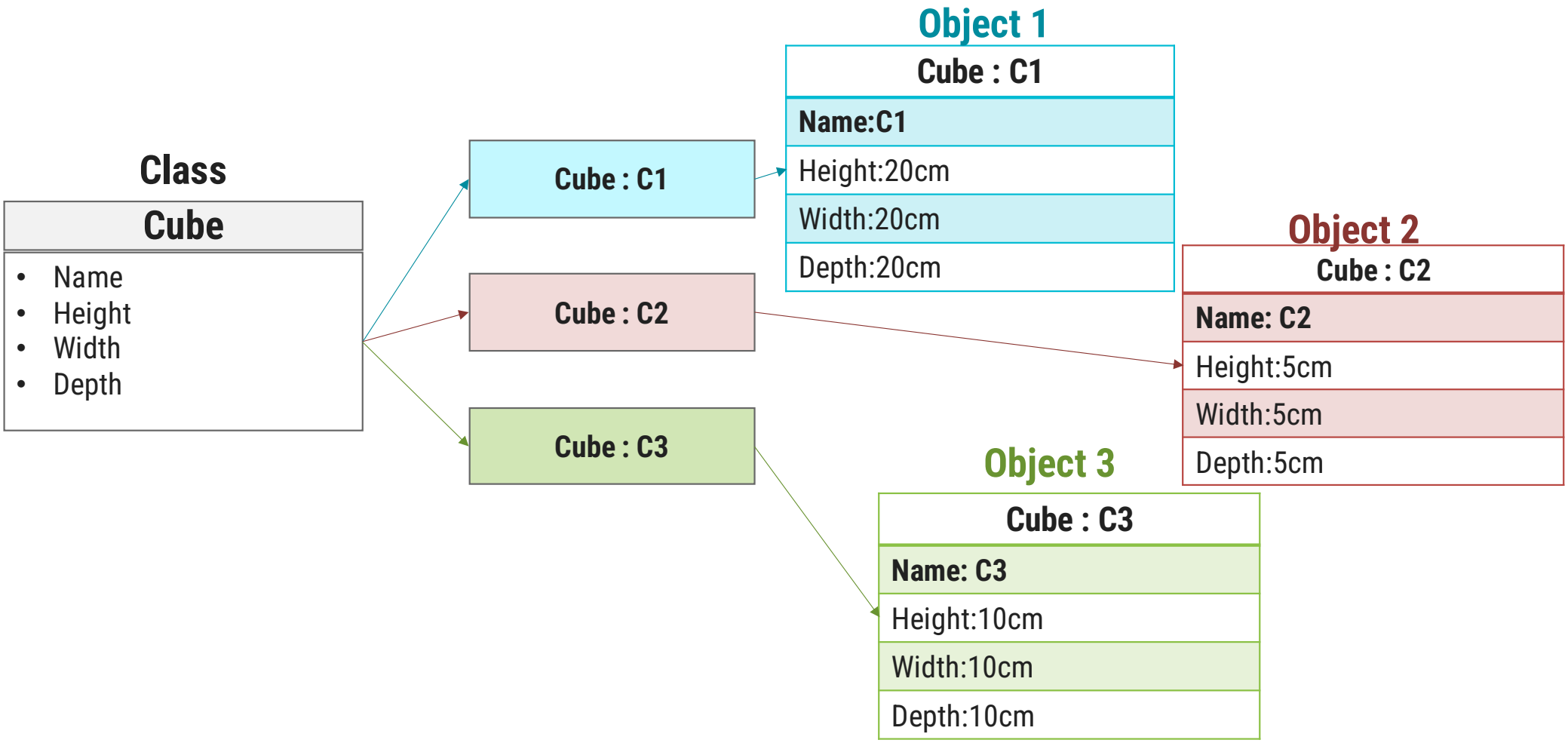
Class and Objects



Class and Objects



Class and Objects



Creating Object & Accessing members

- **new** keyword creates new object
- Syntax:

```
ClassName objName = new ClassName();
```

  Example :

```
SmartPhone iPhone = new SmartPhone();
```

- Object variables and methods can be accessed using the **dot (.)** operator
- Example:

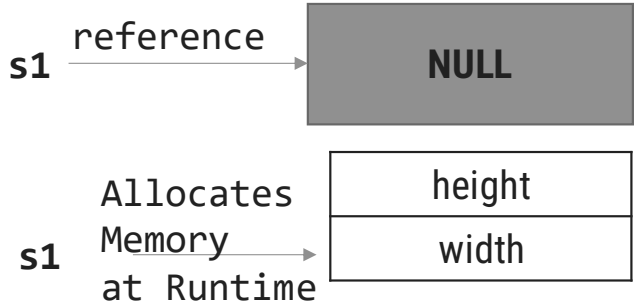
```
iPhone.storage = 8000;
```

Declaring an Object

MyProg.java

```
1. class Square{
2.      double height;
3.      double width;
4. }
5. class MyProg{
6.      public static void main(String[] args) {
7.         Square s1; //declare reference to object
8.         s1= new Square();//allocate a Square object
9.   }
10.}
```

An object reference is similar to a memory pointer.



- **new** operator dynamically allocates memory for an object
- Here, s1 is a variable of the class type.
- The class name followed by parentheses specifies the constructor for the class.
- It is important to understand that **new** allocates memory for an object during run time.

WAP using class Person to display name and age

```
1. class MyProgram {
2. public static void main(String[] args) {
3.   Person p1= new Person();
4.   Person p2= new Person();
5.   p1.name="modi";
6.   p1.age=71;
7.   p2.name="bachchan";
8.   p2.age=80;
9.   System.out.println("p1.name="+p1.name);
10.  System.out.println("p2.name="+p2.name);
11.  System.out.println("p1.age="+p1.age);
12.  System.out.println("p2.age="+p2.age);
13. }//main()
14.}//class myProgram
```

```
15.class Person
16.{
17.   String name;
18.   int age;
19.}//class person
```

Output

```
p1.name=modi
p2.name=bachchan
p1.age=71
p2.age=80
```

WAP using class Person to display name and age with method

```
1. class MyProgram {
2. public static void
        main(String[] args){
3.   Person p1=new Person();
4.   Person p2=new Person();
5.   p1.name="modi";
6.   p1.age=71;
7.   p2.name="bachchan";
8.   p2.age=80;
9.   p1.displayName();
10.  p2.displayName();
11.  p1.displayAge();
12.  p2.displayAge();
13. } //main()
14.} //class myProgram
15.class Person{
16.   String name;
17.   int age;
18.public void displayName(){
19. System.out.println("name="+name);
20. }
21.public void displayAge(){
22.  System.out.println("age="+age);
23. }
24.}//class person
```

Output

```
name=modi
name=bachchan
age=71
age=80
```

WAP using class Rectangle and calculate area using method

```
1. import java.util.*;
2. class MyProgram {
3. public static void main(String[] args){
4.   Rectangle r1=new Rectangle();
5.   Scanner sc=new Scanner(System.in);
6.   System.out.print("enter height:");
7.   r1.height=sc.nextFloat();
8.   System.out.print("enter width:");
9.   r1.width=sc.nextFloat();
10.  r1.calArea();
11. } //main()
12.}//class myProgram
13.class Rectangle{
14.float height;
15.float width;
16.public void calArea()
   {
17.System.out.println(
   "Area="+height*width);
18. } //calArea()
19.} //class
```

Output

```
enter height:30.55
enter width:20.44
Area=624.442
```

WAP using class Rectangle and calculate area with Return value

```
1. import java.util.*;
2. class MyProgram {
3. public static void main(String[] args){
4.   float area;
5.   Rectangle r1=new Rectangle();
6.   Scanner sc=new Scanner(System.in);
7.   System.out.print("enter height:");
8.   r1.height=sc.nextFloat();
9.   System.out.print("enter width:");
10.  r1.width=sc.nextFloat();
11.  area=r1.calArea();
12.  System.out.println("Area="+area);
13. }//main()
14.}//class myProgram
15.class Rectangle{
16.float height;
17.float width;
18.public float calArea()
   {
19.   return height*width;
20. }//calArea()
21.}//class
```

Output

```
enter height:30.55
enter width:20.44
Area=624.442
```

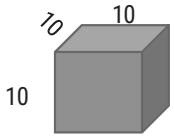
WAP using class Cube and calculate area using method with parameter

```
import java.util.*;
class MyProgramCube {
public static void main
              (String[] args){
  float area;
  Cube c1= new Cube();
  area=c1.calArea(10,10,10);
System.out.println("area="+area);
 }//main()
}//class myProgram
```

```
class Cube{
float height;
float width;
float depth;
float calArea(float h, float
                        w, float d)
{   height=h;
    width=w;
    depth=d;
  return height*width*depth;
 }//calArea()
}//class
```

Output

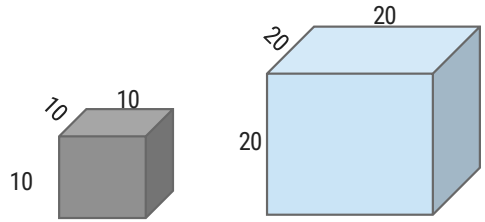
```
area=1000.0
```



WAP using class Cube and calculate area of two objects

```
1. import java.util.*;
2. class MyProgramCube {
3. public static void main
4.                (String[] args){
5.   float area;
6.   Cube c1= new Cube(); //Obj1
7.   Cube c2= new Cube(); //Obj2
8. System.out.println("c1 area="
              +c1.calArea(10,10,10));
9. System.out.println("c2 area="
              +c2.calArea(20,20,20));
10. } //main()
11.} //class
```

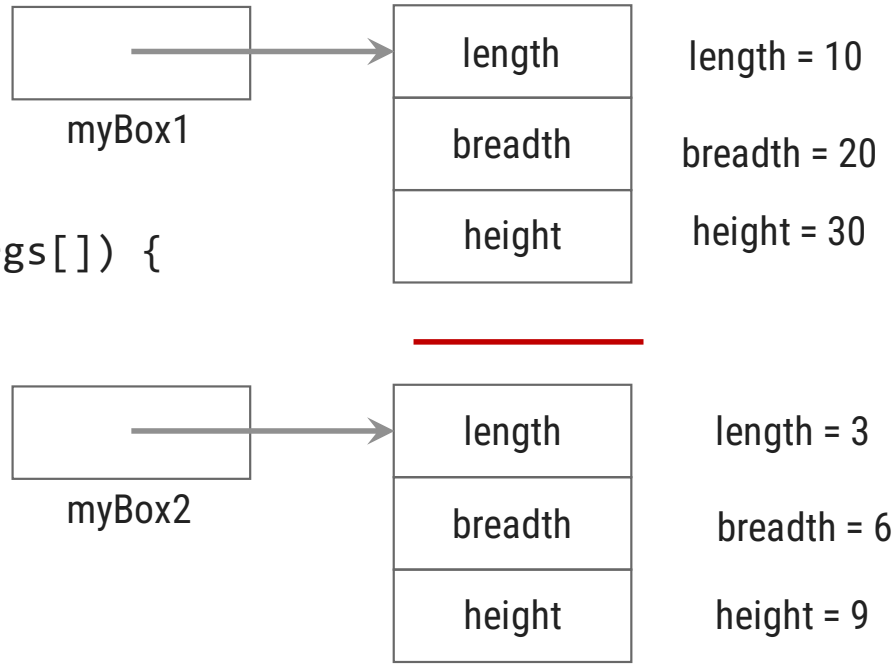
```
12.class Cube{
13.float height;
14.float width;
15.float depth;
16.float calArea(float h, float
                        w, float d)
17.{   height=h;
18.    width=w;
19.    depth=d;
20.   return height*width*depth;
21. } //calArea()
22.} //class
```



Output

```
c1 area=1000.0
c2 area=8000.0
```

```java
class Box {
        double length;
        double breadth;
        double height;
}
class BoxDemo {
        public static void main(String args[]) {
                Box myBox1 = new Box();
                Box myBox2 = new Box();
                double vol;

                myBox1.length = 10;
                myBox1.breadth = 20;
                myBox1.height = 30;

                myBox2.length = 3;
                myBox2.breadth = 6;
                myBox2.height = 9;

                vol = myBox1.length * myBox1.breadth * myBox1.height;
                System.out.println("Volume is " + vol);
                vol = myBox2.length * myBox2.breadth * myBox2.height;
                System.out.println("Volume is " + vol);
        }
}
```



Class vs. Object

Class	Object
Class is a Blueprint or template.	Object is the instance of a class.
Class creates a logical framework that defines the relationship between its members.	When you declare an object of a class, you are creating an instance of that class.
Class is a logical construct.	An object has physical reality. (i.e., an object occupies space in memory.)
Class is a group or collection of similar object. e.g. Car	An Object is defined as real-world entity e.g.:Audi, Volkswagen, Tesla, Ferrari etc.
Class is declared only once	An Object can be created as many times as required

Class doesn't allocate memory when it is created.	An Object allocates the memory upon creation

Class can exist without its objects.	An Object can't exist without a class.

Class: Profession **Class**: Mobile **Class**: Subject **Class**: Student **Class**: Color	**Object**: Doctor, Teacher, Lawyer, Politician… **Object**: iPhone, Samsumg, 1plus… **Object**: Maths, English, Science, Computer… **Object**: John, Aarav, Smita… **Object:** Blue, Green, Red, Yellow, Violet, Black…

Constructor

Constructor

- A constructor in Java is a **special type of method** that is used to initialize objects.
- The constructor is called when an object of a class is created.
- A *constructor* **initializes** an object immediately upon creation.
- It has the same name as the class in which it resides and is syntactically similar to a method.
- JVM first allocates the memory for variables (objects) and then executes the constructor to initialize instance variables.
- The JVM calls it automatically when we create an object.
- A constructor defines what happens when an object of a class is created.

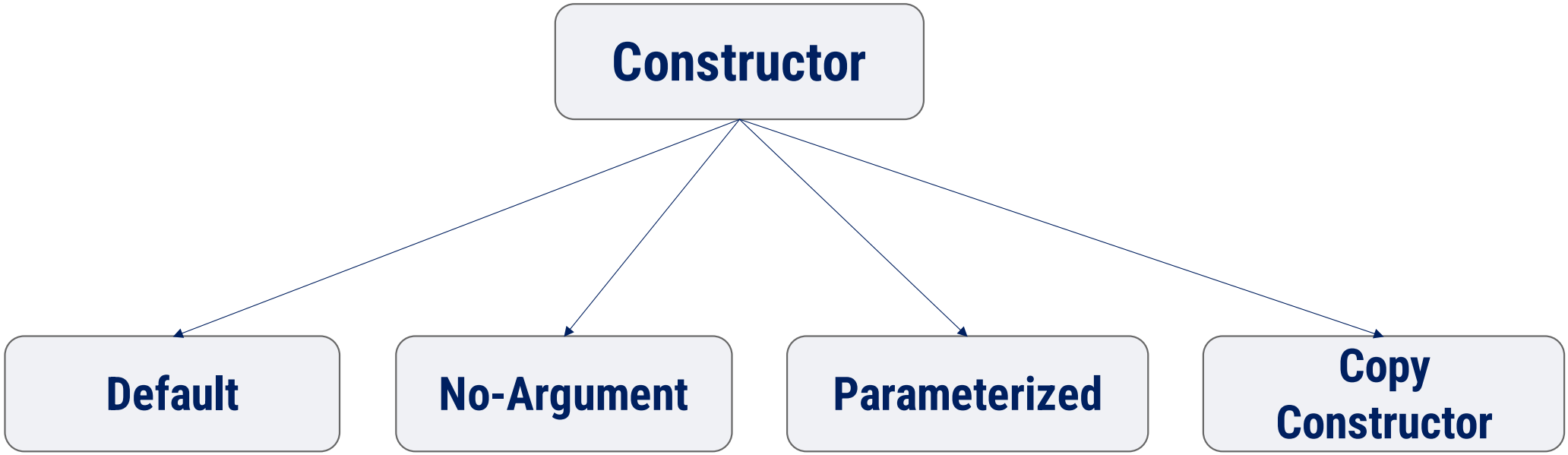
```
1. class Cube{
2. instance variable1
3. instance variable1
4. …
5.    Cube()
6.    {
7.     //initailze
                 object
      }
8. } //class
```

Properties of constructor

- Constructor is *invoked automatically* whenever an object of class is created.
- Constructors do *not* have *return types* and they cannot return values, not even void.
- All classes have *constructors* by *default*: if you do not create a class constructor yourself, Java creates one for you known as **default constructor**.
- Constructor is a method that is called at runtime during the object creation by using the **new** operator. The JVM calls it automatically when we create an object.
- It is called and executed only once per object.
- It means that when an object of a class is created, constructor is called. When we create 2nd object then the constructor is again called during the second time.

```
1. class Demo{
2. instance variable1
3. instance variable1
4. …
5.    Demo()
6.    {
7.     //initailze
                 object
      }
8. } //class
```

Types of Constructor



Default Constructor

Default Constructor: MyConst.java

```
1. class MyConst {
2. public static void main(String[] args)
      {
3.            MyConst c=new MyConst();
4.     }
5. }
```

After Compilation

```
1. class MyConst{
2.  MyConst(){
3.           //Default Constructor…
4.           }
5. public static void main(String[] args)
      {
6.        c=new MyConst();
7.     }
8. }
```

Default Constructor

- Once you define your own constructor, the default constructor is no longer used.
- The default constructor automatically initializes all instance variables to zero.

Example:

```
1. class MyAccount{
2.   int accNo;
3.   double balance;
4.  MyAccount(){
5.     //default constructor
6.   }
7. }//class
8. class MyBank {
9. public static void main(String[] args){
10.  MyAccount ma= new MyAccount();
11.   System.out.println("balance="+ma.balance);
12. }//main()
13.}//class
```

Output

```
balance=0.0
```

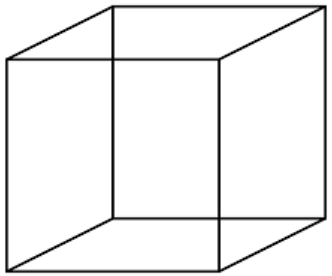
No-Argument Constructor

No-Argument Constructor: MyMain.java

```
1. class Cube {
2. double width;
3. double height;
4. double depth;
5. Cube()
6. {
7.    System.out.println("
          Constructing cube");
8.    width = 10;
9.    height = 10;
10.   depth = 10;
11.  }//Cube()
12.}//class
```

```
13.class MyMain{
14.public static void
   main(String[] args){
15.          Cube c=new Cube();
16.   }
17.}
```

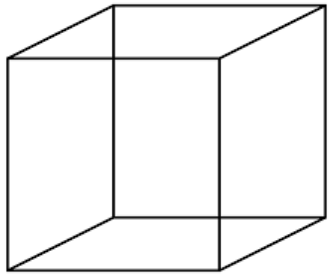
If you implement any constructor then you no longer receiv
*a **default constructor** from Java compiler.*



Parameterized Constructor

Parameterized Constructor: MyMain.java

```
1. class Cube {
2. double width, height, depth;
3. Cube(double w, double h, double d)
4.  { System.out.println("
                    Constructing cube");
5.    width = w;
6.    height = h;
7.    depth = d;
8.   }//Cube()
9. }//class
```

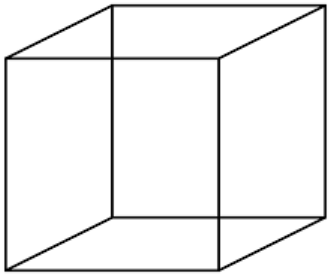


```
10.class MyMain{
11.public static void
            main(String[] args){
12.   Cube c=new Cube(10,10,10);
13.   }
14.}
```

Parameterized Constructor: MyMain.java with method

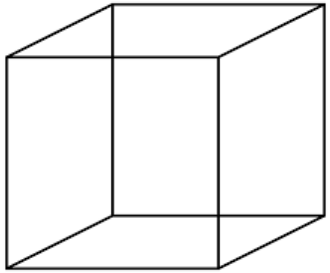
```
1. class Cube {
2. double width,height,depth;
3. Cube(double w, double h, double d)
4.   {System.out.println("Constructing cube");
5.    width = w;
6.    height = h;
7.    depth = d;
8.   }//cube()
9.   void calVolume(){
10.    System.out.println("Volume="
                          +width*height*depth);
11.  } //calVolume()
12.} //class
```

```
13.class MyMain{
14.public static void
       main(String[] args){
15.    Cube c=new Cube(10,10,10);
16.    c.calVolume();
17.    }
18.}
```



Parameterized Constructor: method with return value

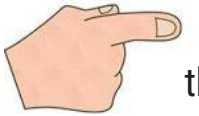
```
1. class Cube {
2. double width,height,depth;
3. Cube(double w, double h, double d)
4.    {System.out.println("Constructing cube");
5.     width = w;
6.     height = h;
7.     depth = d;
8.    }//cube()
9.    double calVolume(){
10.       return width*height*depth:
11.   }//calVolume()
12. }//class
```



```
13. class MyMain{
14. public static void
    main(String[] args){
15.   double vol;
16. Cube c=new Cube(10,10,10);
17.    vol=c.calVolume();
18.    System.out.println("
                    Volume="+vol);
19.    }
20. }
```

Output

```
Constructing cube
Volume=1000.0
```



this

‘*this*’ keyword

```
class Box {
    double length;
    double breadth;
    double height;
    Box(double l, double b, double h) {
        System.out.println("Constructing Box");
        length = l;
        breadth = b;
        height = h;
    }
    void volume() {
        double volume = length * breadth * height;
        System.out.println("Volume is " + volume);
    }
}
class BoxDemo {
    public static void main(String args[]) {
        Box myBox1 = new Box(10,20,30);
        Box myBox2 = new Box(3,6,9);
        myBox1.volume();
        myBox2.volume();
    }
}
```

Creates ambiguity for compiler

length is instance variable as well as length is formal parameter of method

'*this*' Keyword

- *this* is a reference variable that refers to the current object.
- *this* can be used to invoke current object's method.
- *this()* can be used to invoke current class constructor
- *this* can be passed as a parameter to constructor and method call.
- *this* can be used to return the current object from the method.

Copy Constructor

Copy Constructor

- It is a special type of constructor that is used to create a new object using the existing object of a class that had been created previously.
- It creates a new object by initializing the object with the instance of the same class.

```
Constructor 2(Constructor 1)
{
…
}
```

Copy Constructor

Copy Constructor: MyProgramCopy.java

```java
class Student{
    String name;
    int rollno;
    Student(String s_name, int s_roll){
     System.out.println("ConstructorInvoked");
      this.name=s_name;
      this.rollno=s_roll;
    } //Constructor1

    Student(Student s){ //CopyConstructor
     System.out.println("CopyConstructor Invoked");
    this.name=s.name;
    this.rollno=s.rollno;
   } //Constructor2

   public void display(){
    System.out.print("name="+name);
    System.out.println("  rollno="+rollno);
 } // display()
} //class
```

```java
class MyProgramCopy {
public static void main(String[] args){
  float area;
  Student s1=new Student("darshan",101);
  //invoking Copy Constructor
  Student s2=new Student(s1);
  s1.display();
  s2.display();
 } //main()
} //class myProgram
```

Output

```
Constructor Invoked
CopyConstructor Invoked
name=darshan  rollno=101
name=darshan  rollno=101
```

Advantages of Copy Constructor

- It is easier to use when our class contains a complex object with various parameters.
- Whenever we need to add all the field of a class to another object, then just send the reference of previously created object.
- One of the most importance of copy constructors is that there is no need for any typecasting.
- Using a copy constructor, we can have complete control over object creation.
- With Copy Constructor, we can pass object of the class as a parameter(pass by reference).

```
Constructor 2(Constructor 1)
{
…
}
```

Copy Constructor

Constructor Overloading

Constructor Overloading

- In addition to overloading normal methods, you can also overload constructor methods.

```
1. class Balance{
2.     int accNo;
3.     double bal;
4.     Balance(){
5.       System.out.println("inside const1");
6.       bal=0;
7.     }
8.     Balance(double b){
9.       System.out.println("inside const2");
10.      bal=b;
11.    }
12.    Balance(int a,double b){
13.      System.out.println("inside const3");
14.      bal=b;
15.      accNo=a;
16.    }
17.}
```

```
1. class Account{
2.     public static void main(String
                                 args[]){
3.   Balance b1= new Balance();
4.   Balance b2= new Balance(100);
5.   Balance b3=new Balance(1201,10000);
6. System.out.println("b1.bal="+b1.bal);
7. System.out.println("b2.bal="+b2.bal);
8. System.out.println("b3.bal="+b3.bal+
                 "b3.accNo="+b3.accNo);
9.     }
10.}
```

Why Constructor?

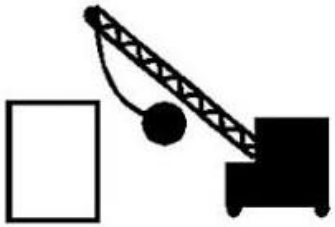
- The pivotal purpose of constructor is to **initialize** the **instance variable** of the class.
- We use constructors to initialize the object with the default or initial state.
- Through constructor, we can request the user of that class for required dependencies.
- A constructor within a class allows constructing the object of the class at runtime.
- Allocates appropriate memory to objects.
- If we need to execute some code at the time of object creation, we can write them inside the constructor.
- **Example:**
  - *If we talk about a Cube class then class variables are width, height and depth.*
  - *But when it comes to creating its object(i.e Cube will now exist in the computer's memory), then can a cube be there with no value defined for its dimensions? The answer is "**NO**".*
  - *So constructors are used to assign values to the class variables at the time of object creation.*

When to use Constructor?

- When we need to execute some code at the time of object creation.
- Used for the initialization of instance variables.
- To assign the default value to instance variables.
- To initializing objects of the class.

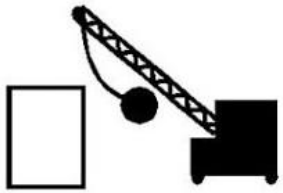
Constructor() vs. Method()

	Constructor()	Method()
Naming	Constructor name must be same as class name.	Method name can be anything.
Return types	Constructor does not have any return type, not even void.	Method must have return types, at least void.
Call	Constructor can be invoked implicitly when object is created.	Method is called by the programmer. Invoked explicitly.
Purpose	To initialize an object	To execute the code
Inheritance	Constructor cannot be inherited by subclass.	Method can be inherited by subclass.



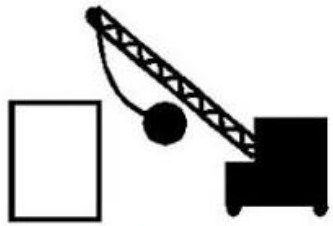
Destructor

Destructor



- **Destructor** is the opposite to the constructor. Constructor is used to initialize objects while the destructor is used to delete or destroy the object that releases the resource occupied by the object.
- **Definition**: *Destructor is an instance member function which is invoked automatically whenever an object is going to be destroyed.*
- In other words, a destructor is the last function that is going to be called before an object is destroyed.
- In java, there is a special method named ***garbage collector*** that automatically called when an object is no longer used.
- When an object completes its life-cycle the garbage collector deletes that object and de-allocates or releases the memory occupied by the object.
- In C++, dynamically allocated objects must be manually released by use of a **delete** operator.
- Java takes a different approach: it handles de-allocation automatically.
- The technique that accomplishes this is known as "***garbage collection***".

Destructor

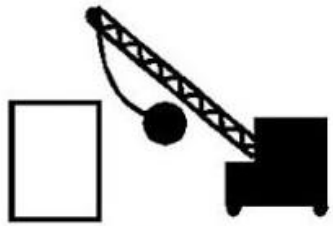


Why Destructor?

- When we create an object of the class(using new), it occupies some space in the memory. If we do not delete these objects, it remains in the memory and occupies unnecessary space.
- To resolve this problem, we use the **destructor**.

- Remember that **there is no concept of destructor in Java**.
- Instead of destructor, Java provides the garbage collector that works the same as the destructor.
- The garbage collector is a program (thread) that runs on the JVM. It automatically deletes the unused objects (objects that are no longer used) and free-up the memory.
- The programmer has no need to manage memory, manually.

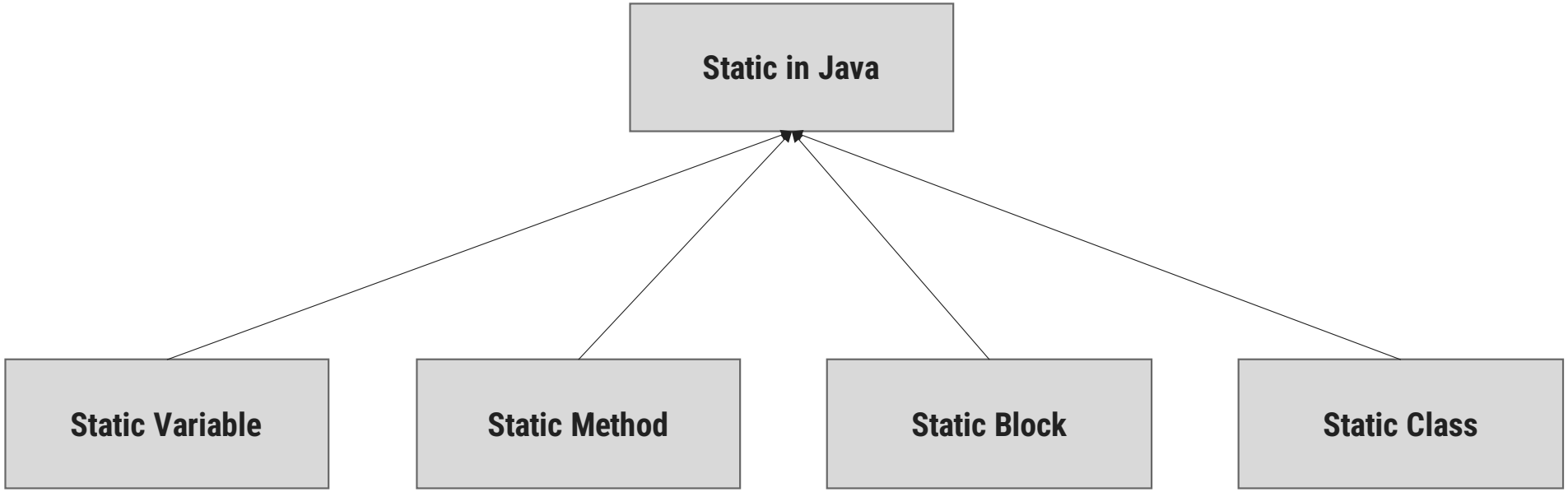
Working of garbage collector(destructor) in java



- When the object is created it occupies the space in the heap. These objects are used by the threads.
- If the objects are no longer used by the thread it becomes eligible for the garbage collection.
- The memory occupied by that object is now available for new objects that are being created.
- When the garbage collector destroys an object, the JRE calls **finalize**() method to close the connections such as database and network connection.

Static

Static in java



Static

- The **static keyword** is used for memory management.
- We can apply static keyword with variables, methods, blocks and nested classes.
- The static keyword belongs to the class than an instance of the class.
- The static can be:
  1. Variable (also known as a class variable)
  2. Method (also known as a class method)
  3. Block
  4. Nested class

Static Variable

- Static variables have a property of preserving their value even after they are out of their scope.
- The static variable gets memory only once in the class area at the time of class loading.
- **Advantage of static variable**: It makes program **memory efficient** (static variable saves memory).
- **Characteristics of static variable:**
  - It is initialized to zero when the first object of its class is created. No other initialization is allowed.
  - Only one copy of that member is created for the entire class and is shared by all the objects of that class, no matter how many objects are created.
  - It is visible only within the class, but its lifetime is the entire program.
  - Static variables are normally used to maintain values common for all objects.
  - The class constructor does not initialize static variable.

Static vs Non-Static Function

MyProgram.java(Non-static function)

```
1. public class MyProgram {
2. public static void
       main(String[] args) {
3.   int a=1,b=2,c;
4.   MyProgram mp=new
                    MyProgram();
5.   c = mp.add(a,b);
6.   System.out.println(c);
   }//main
7.  public int add(int i,int j)
8.  {
9.      return i + j;
10. }
11.}//class
```

MyProgram.java(static function)

```
1. public class MyProgram {
2. public static void
       main(String[] args) {
3. int a=1,b=2,c;
4. c = add(a,b);
5. System.out.println(c);
6. }//main


7. static int add(int i,int j)
8. {           return i + j;
9. }
10.}//class
```

Characteristic of static method

- A static method can call only other static methods and can not call a non-static method from it.
- A static method can be accessed directly by the class name and doesn't need any object
- A static method cannot refer to "this" or "super" keywords in anyway

Static Method: WAP using class Rectangle and calculate area

```
1. import java.util.*;
2. class Rectangle{
3. static float height;
4. static float width;
5. static void calArea() {
6. System.out.println( "Area= "
                    +height*width);
7.  } //calArea()
8. } //class
```

Output

```
enter height:30.55
enter width:20.44
Area=624.442
```

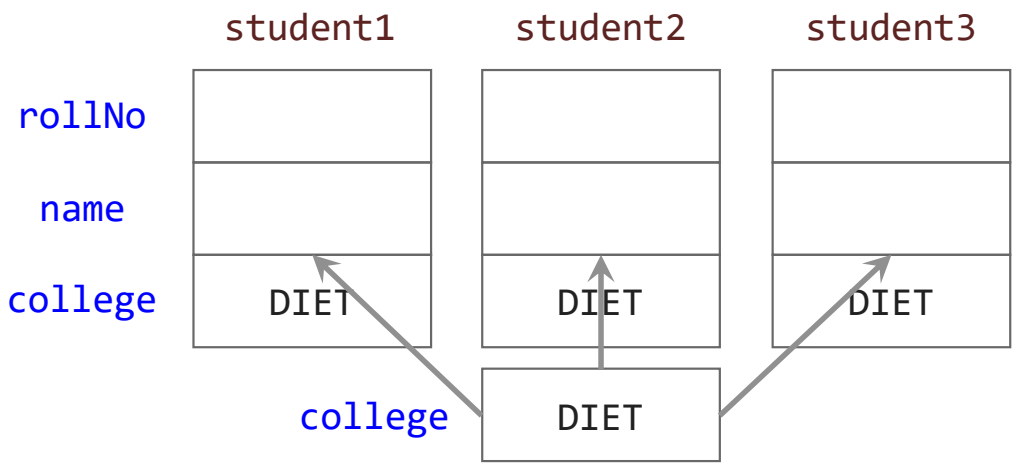
```
13. class MyRectangle {
14. public static void main(String[] args){
15.   Rectangle r1= new Rectangle();
16.   Scanner sc= new Scanner(System.in);
17.   System.out.print("enter height:");
18.   r1.height=sc.nextFloat();
19.   System.out.print("enter width:");
20.   r1.width=sc.nextFloat();
21.   Rectangle.calArea();
22.  } //main()
23. } //class
```

static keyword

- static keyword is mainly used for *memory management*.
- It can be used with
  - Variables
  - Methods
  - Blocks
  - Nested classes
- Basically, static is used for a constant variable or a method that is same for every instance of a class.
- The static variable can be used to refer to the common property of all objects.
- The static variable gets memory only once in the class area at the time of class loading.
- It makes your program memory efficient.
- Syntax

```
static type variablename;
```

```
class Student {
        int rollNo;
        String name;
        String college = "DIET";
        .
        .
        .
}
class StudentDemo {
        public static void main(String args[]) {
                Student student1 = new Student();
                Student student2 = new Student();
                Student student3 = new Student();
                .
                .
                .
        }
}
```



static method

- If you apply static keyword with any method, it is known as static method.
- A static method belongs to the **class** rather than the object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- A static method can access static data member and can change the value of it.
- Restrictions

1. The static method can not use non static data member or call non-static method directly.
2. **this** and **super** cannot be used in static context.

```java
class Student {
    int rollno;
    String name;
    static String college="abc";
    static void change() {
        college = "DIET";
        rollno = 10;
    }
    Student(int r, String n) {
        rollno = r;
        name = n;
    }
    void display() {
        System.out.println(rollno+" "+name+" "+college);
    }
}
class TestStaticMethod {
    public static void main(String args[]) {
        Student.change();
        Student s1 = new Student(111,"Tom");
        Student s2 = new Student(222,"Jerry");
        s1.display();
        s2.display();
       }
}
```

C:\Windows\System32\cmd.exe

```
D:\Java2021Demo>java TestStaticMethod
111 Tom abc
222 Jerry abc
```

We can not use non-static variables in static methods

C:\Windows\System32\cmd.exe

```
D:\Java2021Demo>java TestStaticMethod
111 Tom DIET
222 Jerry DIET
```

Static Block

- Static block is executed exactly once, when the class is first loaded.
- It is used to initialize static variables of the class.
- It will be executed even before the main() method.
- Syntax

```
static {
        //initialisation of static variables…
    }
```

- **How to call static block in java?**
  - ➥ Unlike method, there is no specified way to call a static block.
  - ➥ The static block executes automatically when the class is loaded in memory.

Example: Static Block, Method and Variable

```
1. class StaticDemo {
2.      static int a = 4;      //static variable declared & initialized
3.      static int b;          //static variable declared
4.      static void dispValue(int x) {
5.      System.out.println("Static method initialized.");
6.      System.out.println("x = " + x);
7.      System.out.println("a = " + a);
8.      System.out.println("b = " + b);
9.      } //static method
10.     static {
11.            System.out.println("Static block initialized.");
12.             b= a * 5;
13.     } //static block
14.     public static void main(String args[]) {
15.     System.out.println("inside main()...");
16.     dispValue(44);
17.     } //main()
18.} //class
```

3

1

2

Output

```
Static block initialized.
inside main()...
Static method initialized.
x = 44
a = 4
b = 20
```

Points to remember for static keyword

1. When we declare a field static, exactly a single copy of that field is created and shared among all instances of that class.
2. Static variables belong to a class, we can access them directly using class name. Thus, we don't need any object reference.
3. We can only declare static variables at the class level.
4. We can access static fields without object initialization.
5. Static methods can't be overridden.
6. Abstract methods can't be static.
7. Static methods can't use this or super keywords.
8. Static methods can't access instance variables and instance methods directly. They need some object reference to do so.
9. A class can have multiple *static* blocks.

Mutable and Immutable Objects

- The content of mutable object can be changed, while content of immutable objects can not be changed.

```java
class MutableClass{
    int a;
    void add5() {
        a = a + 5;
    }
}
public class MutableClassDemo {
    public static void main(String[] args) {
        MutableClass m1 = new MutableClass();
        m1.a = 10;
        m1.add5();
        System.out.println(m1.a);
    }
}
```

```java
class ImmutableClass{
    int a;
    int add5() {
        return ( a + 5 );
    }
}
public class MutableClassDemo {
    public static void main(String[] args) {
        ImmutableClass m1 = new ImmutableClass();
        m1.a = 10;
        int ans = m1.add5();
        System.out.println("A in m1 = " + m1.a);
        System.out.println("returned = " + ans);
    }
}
```

Passing Objects as Argument

- In order to understand how and why we need to pass object as an argument in methods, lets see the below example.

```java
class Time{
    int hour;
    int minute;
    int second;
    public Time(int hour, int minute, int second) {
        this.second = second;
        this.minute = minute;
        this.hour = hour;
    }
    void add(Time t) {
        this.second += t.second;
        if(this.second>=60) {
            this.minute++;
            this.second-=60;
        }
        this.minute += t.minute;
        if(this.minute>=60) {
            this.hour++;
            this.minute-=60;
        }
        this.hour += t.hour;
    }
}
```

```java
public class TimeDemo {
    public static void main(String[] args) {
        Time t1 = new Time(11,59,55);
        Time t2 = new Time(0,0,5);

        t1.add(t2);

        System.out.println(t1.hour + ":" +
        t1.minute + ":" + t1.second);
    }
}
```

Array of Objects

- We can create an array of object in java.
- Similar to primitive data type array we can also create and use arrays of derived data types (class).

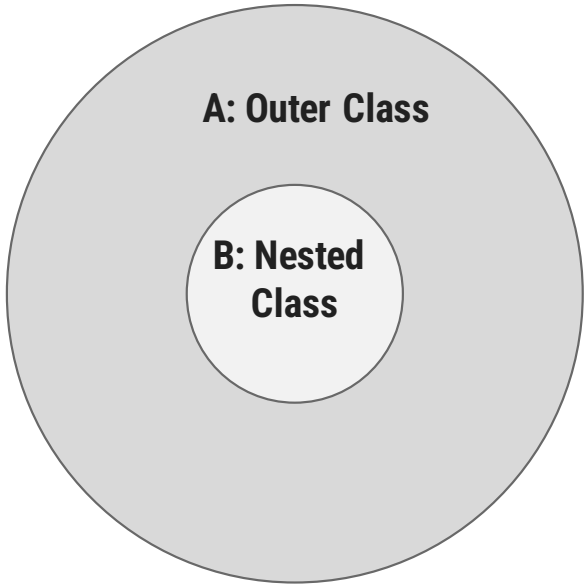
```java
class Student{
    int rollNo;
    String name;

    public Student(int rollNo, String name) {
        this.rollNo = rollNo;
        this.name = name;
    }

    void printStudentDetail() {
        System.out.println("| "+
        rollNo
        +" | -- | "+
        name +" |");
    }
}
```

```java
public class ArrayOfObjectDemo {
    public static void main(String[] args) {
        Student[] stu = new Student[3];

        stu[0] = new Student(101,"darshan");
        stu[1] = new Student(102, "OOP");
        stu[2] = new Student(103,"java");

        stu[0].printStudentDetail();
        stu[1].printStudentDetail();
        stu[2].printStudentDetail();
    }
}
```

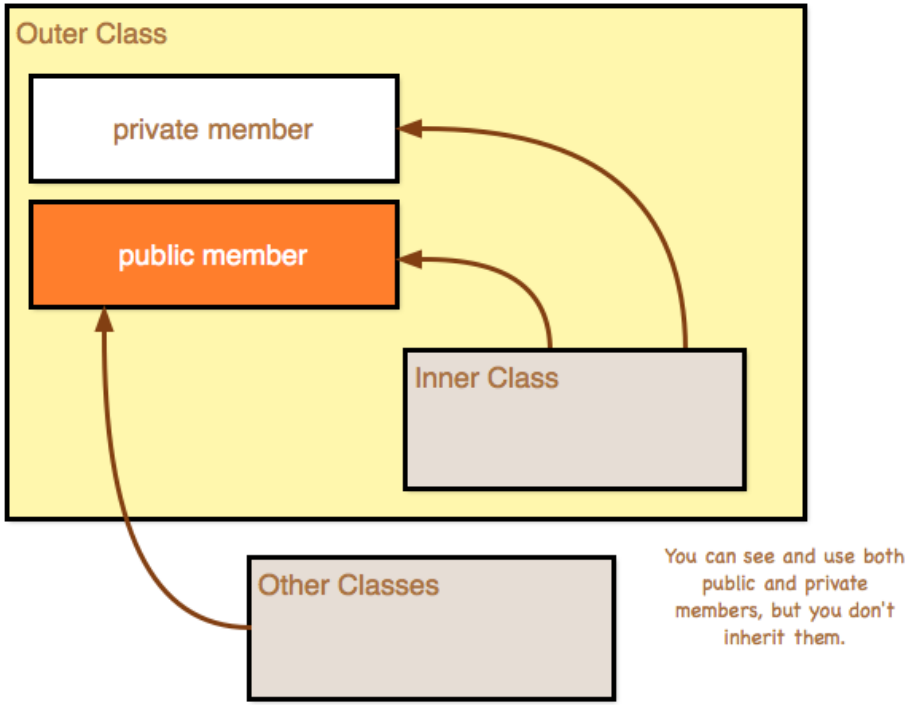
Nested Class

Nested Class

- **Nested Class:** Class within another class
- **Scope**: Nested class is bounded by the scope of its enclosing class.
  - ➥ E.g. class B is defined within class A, then B is known to A, but not outside of A.
- A nested class has access to the members, including private members of the class in which it is nested.
- However, the enclosing class does not have access to the members of the nested class. i.e. Class B can access private member of class A, while reverse is not accessible.

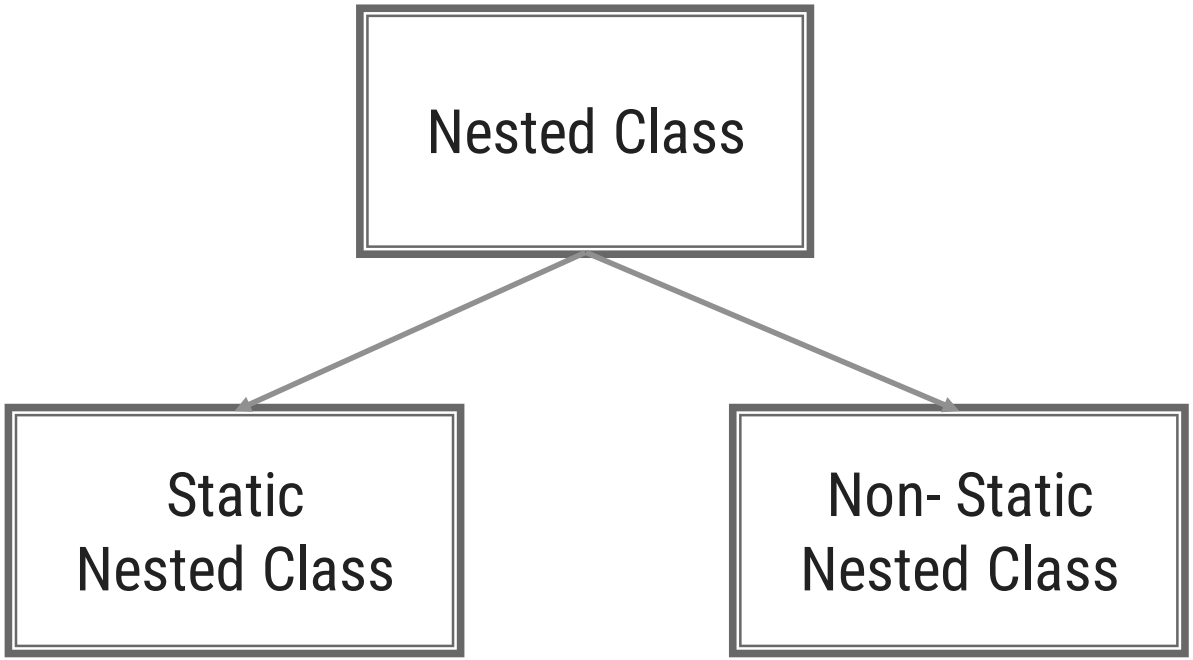


Nested Class



Nested Class

- Types of Nested class:



Non-Static Nested Class: InnerOuterDemo.java

```java
class Outer{
    private int a=100;//instance variable
    void outerMeth(){
      Inner i= new Inner();
      System.out.println("inside outerMeth()...");
      i.innerMeth();
    }
    class Inner{
      int b=20;
      void innerMeth(){
          System.out.println("inside innerMeth()..."
                                          +(a+b));
           }
     } //inner class
} //outer class
```

```java
class InnerOuterDemo{
public static void main(String[]
                                  args)
    {
            Outer o= new Outer();
            o.outerMeth();
     }
} //InnerOuterDemo
```

Output

```
inside outerMeth()...
inside innerMeth()...120
```

Static Nested Class: InnerOuterDemo

```
1.  class Outer{
2.      static int a=100; //a=10
3.      void outerMeth(){
4.        Inner i= new Inner();
5.        System.out.println("inside outerMeth()...");
6.        i.innerMeth();
7.      }
8.      static class Inner{
9.        int b=20;
10.       void innerMeth(){
11.           System.out.println("inside innerMeth()..."
12.                                          +(a+b));
13.            }
14.     } //inner class
15. } //outer class
```

error: non-static variable a cannot be referenced from a static context

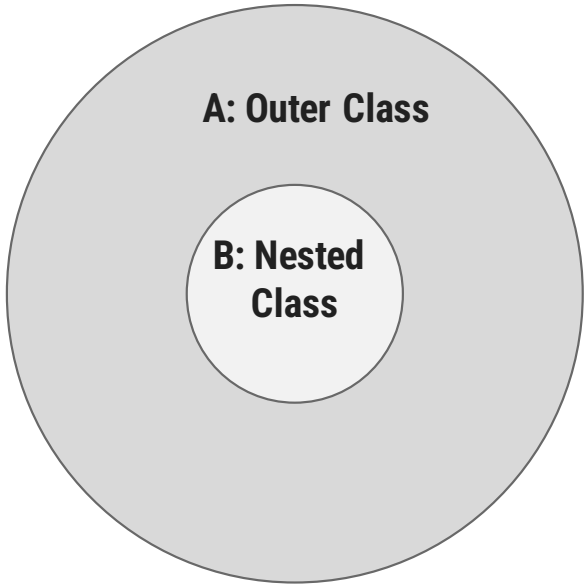
```
16. class InnerOuterDemo{
17. public static void main(String[]
                                  args)
18.     {
19.             Outer o= new Outer();
20.             o.outerMeth();
21.      }
22. } //InnerOuterDemo
```

Output

```
inside outerMeth()...
inside innerMeth()...120
```

Points to remember: Inner class

- Inner class implements **a security mechanism in Java**.
- Reduces encapsulation, more organized code by logically grouping the classes.



Package (Not part of this Unit)

- A **Package** can be defined as a **grouping** of related types providing access protection and name space management.
- Programmers can define their own packages to bundle group of classes/interfaces, etc.
- Packages are used in Java in order to
  1. prevent **naming conflicts**
  2. control **access**,
  3. make **searching/locating** of classes/interfaces easier.
- It is a good practice to group related classes implemented so that a programmer can easily determine that the classes, interfaces are related.

Creating a package

- To create a package you need to write **package** statement **followed** by the **name of the package**.
- Syntax : package package_name;
- Example :

```
package darshan_student;

class Student {
        // code
}
```

- The package statement should be the **first line** in the source file.
- There can be **only one package statement** in each source file, and it applies to all types in the file.
- If a package statement is **not used** then the class/interfaces will be put into an **unnamed package**.

Package (Example)

```java
package myPackage;
public class Animal {
    public String name;
    public void eat(){
        System.out.println("Organic Food !!!!");
    }
    public static void main(String[] args){
        Animal a = new Animal();
        a.eat();
    }
}
```

.

Represent the current directory

- To compile

  javac **-d** **.** Animal.java

- To Run the class file

  java **myPackage.Animal**

import keyword

- **import** keyword is used to import built-in and user-defined packages into your java source file so that your class can refer to a class that is in another package by directly using its name.
- There are 3 different ways to refer to class/interface that is present in different package
  - ➥ import the class/interface you want to use.
  - ➥ import all the classes/interfaces from the package.
  - ➥ Using fully qualified name.
- We can import a class/interface of other package using a import keyword at the first line of code.
- Example :

```
import java.util.Scanner;

public class DemoImport {
    public static void main(String[] args)
    {
        Scanner s = new Scanner(System.in);
        // Code
    }
}
```

import (importing all class/interface)

- We can import all the classes/interfaces of other package using a import keyword at the first line of code with the wildcard (*).
- Example :

```
import java.util.*;
public class DemoImport {
    public static void main(String[] args)
    {
        Scanner s = new Scanner(System.in);
        Date d = new Date();
        // Code
    }
}
```

- It is possible to use classes from other packages without importing the class using fully qualified name of the class.
- Example :

```
java.util.Scanner s = new java.util.Scanner(System.in);
```

Static Import

- The static import feature of Java 5 facilitate the java programmer to **access any static member** of a class **directly**.
- **Advantage** : Less coding is required if you have to access any static member of a class more frequently.
- **Disadvantage** : If you overuse the static import feature, it makes the program unreadable and unmaintainable.

```
import static java.lang.System.out;
public class S2{
  public static void main(String args[]){
        out.println("Hello main");
  }
}
```

We need not to write System.out as we have imported the out statically

Access Control

Modifier	Same Class	Same Package Sub Class	Same Package Non Sub Class	Different Package Sub Class	Different Package Non Sub Class
Private	✓				
Default	✓	✓	✓		
Protected	✓	✓	✓	✓	
Public	✓	✓	✓	✓	✓

Programs

- Write a class named Rectangle to represent a rectangle. It contains following members:
  Data: width (double) and height (double) that specify the width and height of the rectangle.
  Methods:
  1. A no-arg constructor that creates a default rectangle.
  2. A constructor that creates a rectangle with the specified width and height.
  3. A method named getArea() that returns the area of this rectangle.
  4. A method named getPerimeter() that returns the perimeter.
- Write a program to create circle class with area function to find area of circle.
- Define time class with hour and minute. Also define addition method to add two time objects.
- Declare a class called student having following data `members:id_no, no_of_subjects_registered, subject_code, subject_credits, grade_obtained and spi`. Define constructor and `calculate_spi` methods. Define main to instantiate an array for objects of class student to process data of n students.

Thank You